VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2015

Jan Hájovský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Hájovský**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: profiq s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

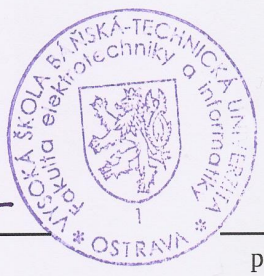
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Konzultant bakalářské práce: Mgr. Gabriel Puhalla

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 1. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. května 2015 .....

Tímto bych rád poděkoval panu Mgr. Gabrielu Puhallovi za umožnění absolvování bakalářské praxe a vedení v jejím průběhu, vedoucímu této práce panu Ing. Martinu Němcovi, Ph.D. za schválení, umožnění vykonání a vedení této praxe a spolupracovníkům ve firmě za rychlé přijetí mezi sebe.

Abstrakt

V práci popíšu absolvování individuální bakalářské praxe ve firmě profiq a mou práci na projektu Frame, která zahrnovala otestování administračního rozhraní a rozhraní pro koncové uživatele. V rámci této práce jsem se seznámil s různými způsoby a technikami testování, díky kterým jsem byl schopen navrhnout testovací případy a provádět manuální vykonávání testů. Manuální testování jsem však postupně nahrazoval automatizovanými testy pomocí Selenium Webdriver a jazyka Python, neobešlo se to však bez problémů, zejména podpory mezi prohlížeči a ověřením WebGL obsahu. Dokument obsahuje také rozbor teoretických a praktických znalostí a dovedností získaných v průběhu studia a uplatněných při práci na tomto projektu a porovnává je se znalostmi získanými v průběhu praxe.

Klíčová slova: testování software, automatizace testů, Selenium Webdriver, Python, WebGL, canvas

Abstract

In the thesis I will describe the process of individual professional practice in the profiq company and my work on the Frame project, which includes testing of admin interface and interface for end users. While working on this task, I have learned different testing types and techniques, which allowed me to design test cases and to perform manual execution of the tests. Manual execuction is gradually replaced by automated tests using the Selenium Webdriver and the programming language Python. A couple of issues occured during the implementation of automation e.g. different browser support or WebGL content verification. There is also a summary of theoretical and practical knowledge gained during the school studies applied while working on the project and a comparison between them and information gained while working on this project.

Keywords: software testing, test automation, Selenium Webdriver, Python, WebGL, canvas

Seznam použitých zkratek a symbolů

AWS	–	Amazon Web Services
CSS	–	Cascading Style Sheets
HTML	–	Hyper Text Markup Language
HTML5	–	Hyper Text Markup Language v. 5
JS	–	JavaScript
LTE	–	Long Term Evolution
OOP	–	objektově orientované programování
PIL	–	Python Imaging Library
RGB	–	červená zelená modrá
SMTP	–	Simple Mail Transfer Protocol
URL	–	Uniform Resource Locator
W3C	–	World Wide Web Consortium
WebGL	–	Web Graphics Library
XML	–	Extensible Markup Language
XPath	–	XML Path Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Absolvoval jsem bakalářskou praxi ve společnosti profiq s.r.o., kde jsem byl zařazen v oblasti testování software. Po nastudování teorie způsobů, technik a kategorií testování jsem byl přiřazen k projektu Frame. Projektu Frame jsem měl otestovat administrační rozhraní a rozhraní pro koncové uživatele. Pro projekt dosud nebyly žádné specifikace k testování, měl jsem tedy začít návrhem testovacích případů, které jsem poté manuálně prováděl. Vykonávání testů mělo být následně automatizováno, k čemuž jsme jako firma měli navrhnout vhodný způsob. Automatizace měla být nasazena proto, aby mohly být testy spouštěny častěji k ověření toho, zda systém pracuje správně, a dále například po nasazení změn, zda-li nedošlo k ochromení předchozí funkčnosti. Při programování automatických testů jsem narazil na několik problémů týkajících se například porovnávání WebGL obsahu, ale také rozdílné podpory nástroje Selenium Webdriver pro různé webové prohlížeče.

2 Zaměření firmy profiq s.r.o. a pracovní zařazení studenta

Firma profiq [1] poskytuje služby softwarového inženýrství z České Republiky pro softwarové společnosti v Silicon Valley, které vyvíjejí software agilním způsobem. Tyto služby zahrnují vývoj, testování a údržbu softwaru.

Služba je poskytována ve více formách. Od té nejjednodušší, kterou je poskytnutí jednotlivých specialistů pro zákazníka, přes komplexnější, kterou je poskytnutí celého týmu, až po kompletní zastřešení dodávky včetně inženýrských zdrojů a managementu.

Expertíza firmy spočívá v oblasti webových a mobilních aplikací, cloudu a bezpečnosti.

2.1 Zákazníci firmy

Firma poskytuje své testovací a vývojové služby například společnosti Forgerock, která se zabývá vývojem kompletního softwarového řešení pro správu identit a přístupů, dále také společnosti Sencha, zabývající se mimojiné vývojem JavaScriptového frameworku a mobilních aplikací. Dalším zákazníkem je Frame, který umožňuje spouštění windows aplikací v cloudu.

2.2 Mé pracovní zařazení

Během vykonávání praxe jsem pracoval ve firmě na pozici software inženýra. Konkrétně jsem se zabýval nově získaným projektem Frame v oddělení testování software. Mým cílem bylo zjistit, zda administrační rozhraní a rozhraní pro koncové uživatele funguje dle očekávání a požadavků, a veškeré zvláštnosti či nefunkční části okamžitě hlásit podpoře Framu.

3 Projekt Frame

Frame je softwarový produkt stejnojmenné společnosti [2], který umožňuje spouštění windows aplikací vzdáleně v cloudu za pomocí webového prohlížeče s technologií WebGL. Aplikace běží vzdáleně na virtualizační platformě Amazon Web Services s operačním systémem Windows Server 2012 R2, ale takovým způsobem, že zákazník se nemusí starat o ruční registraci, nastavení a údržbu virtuálního stroje na stránkách AWS, Frame jej připraví a obsluhuje dle požadavků zákazníka upřesněných jak při koupi produktu, tak i v průběhu používání.

Jednou z výhod produktu Frame je minimalizace hardwarových nároků na koncové stanice uživatele. K plnému použití, nastavení a používání je nutné vlastnit pouze zařízení s libovolným operačním systémem s prohlížečem podporujícím technologii WebGL. Ta je podporována ve všech hlavních prohlížečích, jedná se o Google Chrome, Firefox, Safari, Internet Explorer a Opera. Chromebooky s ChromeOS jsou rovněž plně podporovány. Další výhodou je centralizovaná správa a distribuce aplikací a uživatelských přístupů.

Největší nevýhodou je závislost na neustálém a dostatečně rychlém připojení k internetu, jelikož aplikace jsou spouštěny vzdáleně z cloudu. S rostoucími rychlostmi připojení a pokrytím mobilními sítěmi čtvrté generace (LTE) se míra této nevýhody postupně snižuje.

V době psaní bakalářské práce systém poskytuje dva typy účtů, Platform a Education.

Platform je určen převážně pro větší softwarové firmy, kupříkladu Adobe či Autodesk. Frame se pro tyto firmy hodí z několika důvodů. Mohou poskytnout svůj software k vyzkoušení, aniž by musel uživatel na svém počítači cokoli instalovat. Dále se hodí například pro zaměstnance těchto firem, kdy firmě stačí mít méně výkonné stroje a přitom mohou používat tyto produkty bez hardwarových omezení. Stále ale není nutné na koncové stanice nic instalovat a rovněž aktualizace produktů jsou centralizované pro všechny uživatele.

Education je, jak už název napovídá, určen převážně pro vzdělávací organizace. Zde se Frame skvěle uplatní v situacích, kdy například studenti potřebují pracovat s určitým softwarovým produktem, který nemají běžně k dispozici, a zakoupení a správa licencí mezi studenty by byla příliš komplikovaná a nákladná. Vedoucí této vzdělávací organizace může poskytnout přístup k těmto aplikacím studentům a centrálně spravovat také přístup jednotlivých uživatelů k těmto aplikacím. Opět také odpadá problém s méně výkonnými stroji studentů (k použití například pokročilých modelovacích nástrojů).

4 Seznam úkolů zadaných studentovi

Po nastudování teorie testování jsem byl přiřazen na projekt Frame. Projekt Frame měl být otestován se zaměřením na administrační rozhraní a rozhraní pro koncové uživatele a následně mělo být toto testování automatizováno. Jelikož se projekt nacházel v rané fázi, nebyl u něj k dispozici žádný materiál, který by testování a celkově mou práci usnadnil. Má práce zahrnovala:

- seznámení se s projektem
- návrh testovacích případů
- manuální testování
- automatizaci testů

Zákazník kromě přihlašovacích údajů a znalostní báze, tzv. "Knowledge base", ve které se zákazníci můžou dovědět o nových funkcích systému, neposkytl žádné další informace, a já jsem dostal za úkol seznámit se se systémem kompletně sám za účelem zjištění, zda je systém pochopitelný i pro osoby, jež o něm dosud nic neví.

4.1 Vyjádření časové náročnosti

Na seznámení se s projektem bylo vyhrazeno 5 dní, stejně tak na návrh testovacích případů. K manuálnímu testování byl určen půlden každého týdne na praxi. Pokud však byly přidány nové funkcionality či opravy stávajících chyb, bylo návrhu testovacích případů a manuálnímu testování přidáno časových prostředků dle potřeby. Zbytek času byl věnován návrhu a realizaci automatizaci testů.

5 Teorie a manuální testování

Než jsem se vůbec mohl začít věnovat samotným úkolům, musel jsem nastudovat teorii a způsoby testování [3], abych lépe pochopil, co se ode mě bude očekávat a na co se zaměřit.

5.1 Kategorie a typy testování

Testování se může dělit do kategorií, kdo testování provádí a na jaké úrovni se vyvíjený produkt nachází na:

- Testování komponent
- Integrační testování
- Systémové testování
- Akceptační testování

Testováním komponent /též známé jako jednotkové testování/ se rozumí testování jednotlivých samostatných částí, modulů a tříd aplikace, které jsou testovatelné samostatně. Toto testování obvykle provádí programátor.

Integrační testování v sobě zahrnuje testování komunikace mezi jednotlivými komponentami jak v rámci aplikace, tak i s různými částmi systému jako je operační systém, souborový systém apod. Od této úrovně testování nejčastěji provádí dedikovaný specialista, tester.

Systémové testování slouží k ověření systémových vlastností produktu jako jsou výkonnost, bezpečnost, kompatibilita, dostupnost a jiné. Systémové testování poskytne nejvyšší hodnotu v případě, že se testovací prostředí co nejvíce podobá prostředí, ve kterém bude produkt používat zákazník.

Akceptační testování provádí již samotný zákazník či budoucí uživatelé systému na svém testovacím prostředí za účelem ověření, zda aplikace splňuje očekávání zákazníka.

Testování se dá také rozdělit podle typu testování na:

- Funkcionální testování
- Nefunkcionální testování
- Strukturální testování
- Konfirmační a regresní testování

Funkcionální testování /též testování černé skříňky/ zkoumá funkcionality, vlastnosti a spolupráci jednotlivých komponent, ať už dokumentovaných či nikoli, případně celého systému. Patří zde také například testování bezpečnosti.

Nefunkcionální testování v sobě zahrnuje testování výkonu, zátěžové testování, spolehlivost a jiné.

Strukturální testování /též testování bílé skříňky/ umožňuje určit, zda jsou pokryty všechny možnosti, a to analýzou struktury kódu.

Konfirmační a regresní testování má za cíl zjistit, zda-li nahlášená chyba byla opravdu opravena a její oprava nerozbila stávající funkcionalitu.

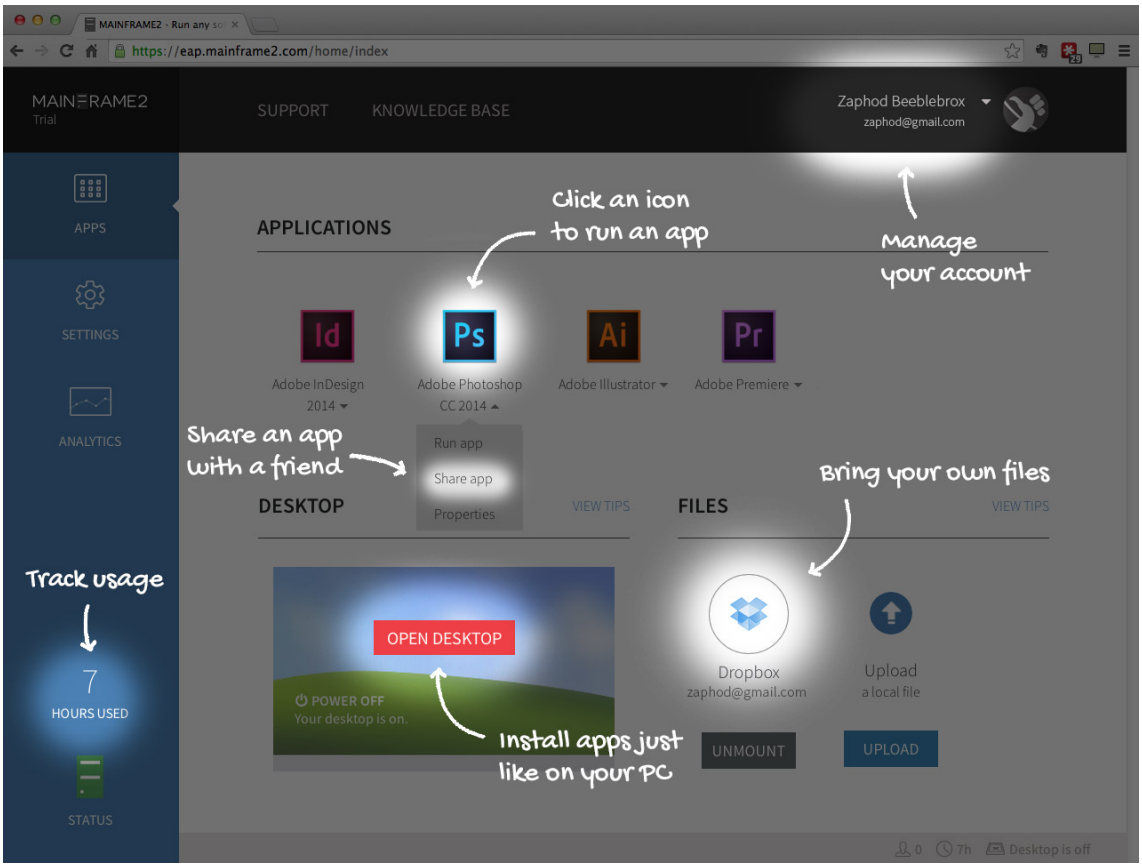
Kategorií, typů a způsobů testování je mnoho, rád bych však ještě zmínil jednu techniku, kterou jsem také využíval.

Průzkumné testování je technika, při které probíhá návrh testů, jejich vykonávání a zaznamenávání zároveň. Průzkumné testování také dovoluje produkt prozkoumat a otestovat i případy, které automatizace nebo specifikace neobsahuje.

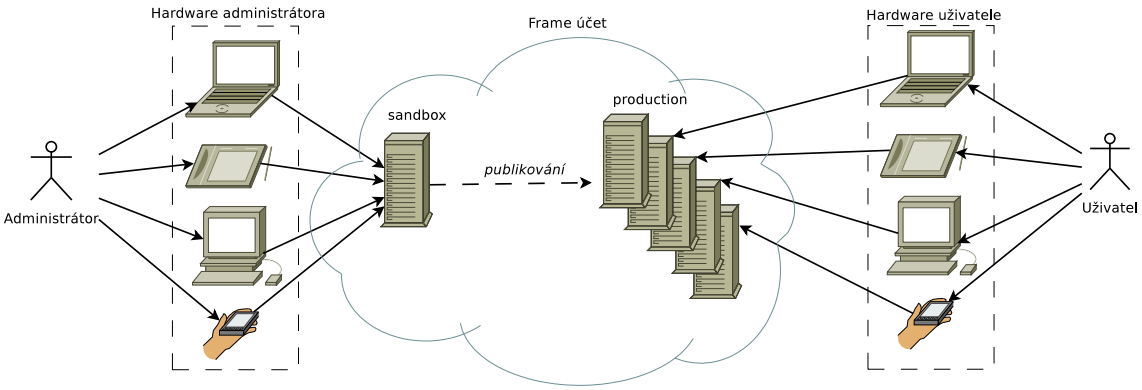
V rámci absolvování bakalářské praxe jsem prováděl nejprve průzkumné, systémové funkcionální testování. Výjimečně jsem prováděl také integrační funkcionální testování, kdy jsem dostal pokyn od Framu, na kterou komponentu systému se zaměřit. Po opravě chyb jsem prováděl konfirmační a regresní funkcionální testování.

5.2 Seznámení se s projektem

Při zakoupení dostane zákazník administrátorský účet na stránkách Frame, viz obrázek 1 na straně 11. Administrátor má k dispozici plně funkční prostředí nazývané sandbox či desktop s operačním systémem Windows Server 2012, na které může nainstalovat libovolnou aplikaci. Následně administrátor tyto aplikace tzv. onboardem připraví ke zveřejnění, a to tak, že z běžícího sandboxu vybere danou aplikaci a přes její kontextovou nabídku zvolí "Onboard to Frame". Pak administrátor vidí tuto aplikaci v přehledu aplikací se značkou, že se jedná o nezveřejněnou aplikaci, a může spustit proces publikování, kdy se tyto aplikace překopírují na tzv. production instance, viz schéma 2 na straně 11. Tyto aplikace pak fungují na kompletně oddělených prostředích, aby se jednotliví uživatelé navzájem neovlivňovali. Jednotliví uživatelé mohou mít, ale nepotřebují žádný účet, aplikace mohou spustit i z html stránky, a to tak, že uživatel buď obdrží od administrátora odkaz získaný ze systému, nebo administrátor může aplikaci (respektive její spouštěč) vložit do libovolné HTML stránky. Pokud uživatelé mají účet, vidí po přihlášení do svého profilu seznam všech dostupných aplikací a nepotřebují nadále jednotlivé odkazy či stránky s vloženými spouštěči.



Obrázek 1: Administrační rozhraní



Obrázek 2: Schéma účtu

5.3 Návrh testovacích případů

Při provádění průzkumného testování jsem zároveň navrhoval testovací případy. Každý testovací případ je popsán prerekvizitami před spuštěním testu, jednotlivými kroky uživatelské interakce a očekávaným chováním s výsledky. Příklad takovýchto testovacích případů můžete vidět v tabulkách 1 a 2 na straně 13. Tyto návrhy testovacích případů byly verifikovány Framem, zda-li došlo ke správnému pochopení systému, jeho chování a očekávaných výsledků.

Tuto fázi jsem opakoval vždy při přidání nové funkčnosti do systému, například nahrávání souborů při běžící aplikaci, která byla přidána až poté, co jsem měl dosavadní funkčnosti systému zmapovány a testovací případy navrženy. V průběhu průzkumného testování jsem často našel nejvíce problémů, jednalo se totiž nejčastěji o nové funkčnosti, kde je větší pravděpodobnost neošetření některých netriviálních krajních případů. Jeden takový netriviální testovací případ ukazuje tabulka 3 na straně 14, která popisuje testovací případ, kdy je proces nahrávání souboru zrušen a vzápětí spuštěn znova. Zde byla nalezena v minulosti chyba, že opakovaný proces nahrávání byl vždy neúspěšný.

Tabulka 1: Nastartování sandbox systému

Testovací případ	Test č.102 - Nastartování sandbox systému
Předpoklad	1. Otevřený internetový prohlížeč na adrese administračního rozhraní 2. Administrátor je do systému přihlášen 3. Sandbox stroj je vypnut
Testovací procedura	**Očekávaný výsledek**
1. Nastartování sandbox stroje stiskem tlačítka "Power on"	Administrátor vidí ukazatel průběhu s odpovídající hláškou. Startování má definováno časový limit 5 minut. V jejich rámci se rovněž očekává, že administrátor vidí informaci o úspěšném nastartování sandboxu.

Tabulka 2: "Onboard"aplikace notepad

Testovací případ	Test č.214 - "Onboard"aplikace notepad
Předpoklad	1. Otevřený internetový prohlížeč na adrese administračního rozhraní 2. Administrátor je do systému přihlášen 3. Sandbox stroj je zapnut
Testovací procedura	**Očekávaný výsledek**
1. Spuštění sandbox stroje stiskem tlačítka "Open desktop"	Spustí se iframe s HTML5 video přehrávačem, ve kterém sandbox nastartoval.
2. "Onboard"systémové aplikace `notepad.exe` otevřením příslušného adresáře, kliknutí pravým tlačítkem myši na soubor aplikace a zvolení volby "Onboard to Frame"	Vyvolá se HTML formulář, ve kterém administrátor vidí ikonu a název aplikace, kterou vybral k "Onboard"u.
3. Potvrzení "Onboard"formuláře stiskem tlačítka "Onboard"	"Onboard"formulář zmizí.
4. Odpojení od sandbox stroje stiskem tlačítka "Disconnect"ve "Start menu"	Iframe s HTML5 video přehrávačem je ukončen. Administrátor vidí seznam "Onboard"ovaných aplikací, ve kterém je navíc oproti stavu před krokem 1) přidána aplikace "notepad".

Tabulka 3: Nahrávání souboru po předchozím zrušeném pokusu

Testovací případ	Test č.1054 - Nahrávání souboru po předchozím zrušeném pokusu
Předpoklad	1. Otevřený internetový prohlížeč na adrese administračního rozhraní 2. Administrátor je do systému přihlášen 3. Produkční instance zapnuta
Testovací procedura	**Očekávaný výsledek**
1. Nastartování aplikace	Aplikace se nastartuje, ale uživatel ji může používat.
2. Zvolení nahrání souboru	Zvolený soubor se začne nahrávat na vzdálenou instanci a uživatel vidí ukazetel průběhu.
3. Zrušení nahrávání	Proces nahrávání se ukončí.
4. Zvolení nahrání souboru	Zvolený soubor se začne nahrávat na vzdálenou instanci a uživatel vidí ukazetel průběhu.

5.4 Manuální testování

Po navržení a schválení testovacích případů bylo nutné určit, které jsou důležité a musí se pravidelně testovat. Než bude totiž automatizace vyvinuta a nasazena, je nutné zajistit průběžné manuální testování. Z navržených případů zákazník identifikoval sadu testů, které otestují funkcionalitu všech částí systému. Tyto testy jsem pravidelně vykonával, výsledky vždy zpracoval a odeslal zákazníkovi ve formátu tabulky, viz obrázek 4 na straně 15. Tímto způsobem jsem byl schopen snadno zjistit problémy se zpětnou kompatibilitou, a díky pravidelným výsledkům jsem měl možnost porovnání s předchozími výsledky a zjištění, kdy se daná chyba poprvé objevila.

Každou chybu jsem hlásil jako nový úkol do systému Asana, což je webová služba určená k organizaci úkolů. Při hlášení každé chyby jsem sepsal kroky k reprodukci dané chyby a také nahrál obrázek či video, na kterém jsem chybu ukázal.

Manuální testování jsem prováděl na různých webových prohlížečích, v pořadí dle priorit zákazníka se jednalo o Chrome, Firefox, Safari a Internet Explorer. K provedení testování jsem musel použít více operačních systémů (Windows 8.1 a OS X 10.10), jelikož poslední verze Safari nelze spustit na OS Windows a naopak zase Internet Explorer nelze nativně spustit na OS X.

Tabulka 4: Výsledky manuálních testů

Detailed status			
Test #	Frame End User	Firefox 33	Comment
Login screen			
500	Login correct	PASSED	
501	Login incorrect	PASSED	
502	Login empty	PASSED	
503	Login with expired account	PASSED	
505	Password recover	N/A	It seems not implemented yet
Application list			
520	Application list shows	PASSED	
521	Upload setup files using file dialog works	PASSED	
522	Upload documents using file dialog works	N/A	This option has been disabled
523	Upload setup files using drag & drop works	PASSED	
524	Upload documents using drag & drop works	N/A	This option has been disabled
525	Start sandbox	PASSED	
526	Share sandbox	FAILED	link to reported issue in Asana
527	Stop sandbox	PASSED	
528	Open the application in sandbox	PASSED	
...	...	...	...

6 Automatizace testů

Dlouhodobým cílem je nahrazení manuálního testování automatizovanými testy za účelem snížení spotřeby lidského času, nutného k otestování produktu. Automatizované testy umožňují mimo jiné častější vykonávání testů, které je navíc nezávislé na rozdílných geografických polohách vývojového a testovacího týmu. Vývojový tým tedy může spustit testování dle potřeby i při nepřítomnosti testovacího týmu.

6.1 Nalezené možnosti

Není možné začít vyvíjet automatizaci bez zvolení vhodného nástroje a přístupu. Jelikož se jedná o webovou aplikaci, je nutné zvažovat nástroje, které webové stránky umí načíst a zpracovat.

Od managera projektu jsem dostal tip na nástroj Selenium Webdriver [4]. Tento open source nástroj umožňuje vytvoření nezávislého okna webového prohlížeče a poskytuje kompletní rozhraní k jeho ovládaní. Jeho největší výhodu vidím ve zpracování JavaScriptu. Při použití Selenia je zajištěno, že chování bude stále stejné a bude vypadat tak, jak ho uvidí i koncový uživatel. Naopak největší nevýhoda tohoto nástroje je hardwarová náročnost, kdy musí běžet grafické prostředí, aby v něm mohl být korektně spuštěn a používán testovaný prohlížeč. Nepříjemná je také rozdílná podpora pro webové prohlížeče, kdy pro každý prohlížeč (vyjma standardně podporovaného Firefoxu) musí být použit speciální soubor, tzv. driver, který zprostředkovává komunikaci mezi programátorovým kódem a prohlížečem. Tento driver je pro každý prohlížeč napsán mírně odlišným způsobem a je v různém stádiu vývoje, a proto má každý jiná omezení (více o těchto omezeních v kapitole 6.6 na straně 22). Mezi oficiálně podporovanými jazyky pro použití se Selenium Webdriver je Java, C#, Ruby, Python a Javascript, mezi neoficiálními je například také Perl, PHP či Objective-C.

Mezi dalšími zvažovanými nástroji bylo použití HtmlUnit pro Javu [5] či urllib2 knihovna pro Python [6]. Tyto nástroje umožní interakci s webovými stránkami bez nutnosti otevření okna prohlížeče, fungují tedy rychleji a s menšími hardwarovými nároky. Nevýhoda je však ve zpracování JavaScriptu, kdy jsou podporovány pouze základní operace nebo dokonce žádné, což je u webové aplikace hojně využívající HTML5 kritická vlastnost. Z tohoto důvodu byly tyto nástroje zamítnuty.

Dalším nalezeným nástrojem byl Watir-Webdriver [7], který, jak jsem zjistil, je postaven také na technologii Selenium Webdriver, avšak poskytuje pouze rozhraní pro programovací jazyk Ruby.

Posledním analyzovaným nástrojem byl komerční softwarový produkt Squish společnosti Froglogic [8], který pracuje na podobném principu jako Selenium Webdriver, přidává však možnost nahrávání testů. Nahrávání testů se ale ukázalo jako nevhodné, v pří-

padě, kdy hodnoty některých atributů jsou statické a jiné zase generovány automaticky. Výsledek automatického nahrávání bych tedy musel vždy projít a upravit tak, aby využívalo neměnné hodnoty.

6.2 Zvolený způsob

Pro automatizaci prohlížení webu a uživatelských interakcí jsem z výše uvedených důvodů zvolil použití Selenium Webdriver. Jak již bylo řečeno, vyniká ve zpracování JavaScriptu, má ale rozdílnou podporu napříč prohlížeči. Od Framu jsme však dostali jasnou prioritu – Google Chrome a Firefox musí být při automatizaci funkční, funkčnost testů v Safari a Internet Explorer by byl příjemný bonus, ale vyžadován není.

Jako jazyk, ve kterém bude automatizace napsána, jsem zvolil Python. Jedná se o multiplatformní skriptovací jazyk, který se dále vyznačuje například dynamickou typovostí či částečnou nutností programátora dodržovat konvence, jelikož například úroveň odsazení je použita jako uvození bloku kódu. Nevýhoda Pythonu, a obecně skriptovacích jazyků, je v mírně vyšší paměťové náročnosti a pomalejším běhu, v mém případě se však nejedná o situaci, kdy by to převážilo uvedené výhody.

6.3 Průběh automatizace

Automatizace není jen o zvolených technologiích, ale také o jejich spojení. V počátcích jsem se snažil zautomatizovat pouze základní případ užití, konkrétně nastartování aplikace, ověření startu a vypnutí aplikace. Tento test jsem prezentoval Framu a společně jsme se rozhodli, že je automatizace vhodná a dostatečně efektivní, a čas investovaný do automatizace se začal zvyšovat, stejně jako složitost a počet scénářů k automatizaci. Z toho důvodu bylo nutné vymyslet strukturu, která mi umožní spouštět libovolné a oddělené sady testů.

Ve fázi, kterou jsem v době psaní této práce využíval, se automatizace testů skládala z několika souborů:

Knihovna Cílem této knihovny je automatizace základních kroků a operací prováděných nad testovaným systémem. Jedná se například o způsoby otevírání kontextových nabídek podle názvu aplikace, přepínání záložek apod. Z těchto jednotlivých operací je následně poskládán testovací případ.

Kód sady testů Obsahuje definice testů a operací, které jsou provedeny. Pro každý test je vyhrazena jedna funkce, jejíž návratová hodnota značí úspěch (True) či neúspěch (False), a to návratovou hodnotou uvedenou v závorce.

Spouštěč sady testů V tomto souboru jsou použity funkce definované v kódu sady testů. Každý test má v tomto souboru vyhrazen svůj řádek, který může být vložen vícekrát či jinak parametrizován, a test tedy poběží vícekrát. Zde je definováno i stan-

dardní chování po skončení testu, obsahující vyčištění, třeba znovunačtení výchozí URL adresy.

Konfigurační soubor Konfigurační soubor v sobě zahrnuje jak specifické proměnné pro každou sadu testů (na základě použité předpony před danou proměnnou), tak globální proměnné jako údaje pro SMTP server k odeslání záznamu v případě neúspěšného testu.

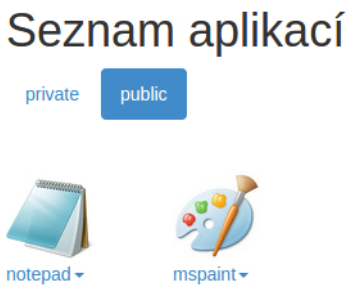
Záznamová třída Jedná se o jednoduchou statickou třídu, která poskytuje operace jako vrácení všech záznamů z neúspěšných testů

Odesílač emailů V případě, že nějaký test selže, může být záhodno informovat support tým, aby situaci ověřil a zareagoval co nejdřív, a aby chybu, pokud možno, neobjevil zákazník.

Porovnávač obrázků Obsahuje funkce k získání procentuální shody dvou obrázků. Nutnost těchto funkcí popíši v sekci 7 na straně 24.

6.4 Ukázka kódu

Mám například HTML stránku z obrázku 3 se strukturou z výpisu 1 na straně 19 a potřebuji lokalizovat element `li` podle popisku aplikace, a zároveň pouze mezi aplikacemi aktuálně vybrané záložky.



Obrázek 3: Testovací HTML stránka

```
<ul id="tabs">
  <li>private<li>
  <li class="active">public<li>
</ul>
<div id="loading" style="display: none;">...</div>
<div id="private" style="display: block;">
  <ul class="app-list">
    <li>
      <a class="title"><img class="icon" />notepad</a>
      <a class="submenu"><ul><li>...</li></ul></a>
    </li>
    <li>
      <a class="title"><img class="icon" />wordpad</a>
      <a class="submenu"><ul><li>...</li></ul></a>
    </li>
  </ul>
</div>
<div id="public" style="display: none;">
  <ul class="app-list">
    <li>
      <a class="title"><img class="icon" />notepad</a>
      <a class="submenu"><ul><li>...</li></ul></a>
    </li>
    <li>
      <a class="title"><img class="icon" />mspaint</a>
      <a class="submenu"><ul><li>...</li></ul></a>
    </li>
  </ul>
</div>
```

Výpis 1: Testovací HTML kód

K tomu může sloužit funkce z výpisu 2 na straně 20. Než začnu záložky a aplikace prohledávat, je nutné počkat na ukončení načítání, k čemuž používám funkci `WebdriverWait`. Tato funkce zajistí čekání, dokud element `div#loading` nepřestane být zobrazený, maximálně však 20 sekund. Pokud je `div#loading` stále viditelný, uloží zprávu do logu /záznamová třída/ a vrací neúspěch False. Další krok pozitivního průběhu je zjištění popisku aktivní záložky, jelikož následně budu prohledávat pouze `div`, jehož `id` je stejné jako popisek aktuální záložky. Pokud by tento krok nebyl poveden, a byla by hledána například aplikace notepad, která je v obou sekcích, nebylo by jednoduše možné rozlišit, do jaké záložky patří. Po zjištění popisku můžeme css selektorem získat seznam všech odpovídajících aplikací. Je nutné si rovněž uvědomit, že jednotlivé položky seznamu aplikací v sobě obsahují další podmenu, které rovněž obsahuje elementy `li`. Proto je nutné v CSS selektoru znakem < omezit hledání pouze na elementy `li`, které jsou přímo vnořené v elementu `ul.apps-list`. Pak už nezbývá než projít

seznamem a kontrolovat text elementu `a.title`, a pokud odpovídá hledanému názvu, položka seznamu pro hledanou aplikaci je vrácena, a programátor může pracovat jak s titulkou aplikace, tak s vnořeným podmenu (otevřít, přesměrovat se a jiné). Pokud žádná aplikace neodpovídá hledané, je uložena zpráva do logu a funkce vrací `False` značící neúspěch.

```
def application_find(self, app_name):
  # wait until application loads
  try:
    WebDriverWait(self.browser, 20).until_not(self.browser.
        find_element_by_css_selector('div#loading').is_displayed()
        )
  except TimeoutException:
    Log.insert('{0} ERROR: Loading applications got stuck!'.
        format(self.act_time()))
    return False
  # find out chosen tab
  act_type = self.browser.find_element_by_css_selector('ul#tabs
      li.active').text
  # find all application in actual tab
  apps = self.browser.find_elements_by_css_selector('div#{0} ul.
      app-list > li'.format(act_type))

  # loop through array of applications
  for li in apps:
    if li.find_element_by_css_selector('a.title').text ==
        app_name:
      return li
  else:
    Log.insert('{0} ERROR: The application "{1}" has not been
        found!'.format(self.act_time(), app_name))
    return False
```

Výpis 2: Vyhledání aplikace v seznamu

Příkladem složitější funkce může být například startovací sekvence aplikace. Zde je více testovaných situací a potenciálních problémů, které musí test odhalit. Při startování je nutné ověřit:

- Spustí-li se okno s aplikací
- Zobrazí-li se startovací sekvence
- Zda-li je časovač správně inicializován
- Pokud je vzdálenost uživatele od serveru dlouhá, zobrazilo-li se varování o vzdálenosti

- Pokud je připojen Dropbox, zda-li došlo k jeho automatickému připojení
- Zda-li obsah aplikace je správný

Z funkcí knihovny dále skládám daný testovací případ. To můžete vidět v ukázce 3, kde daná funkce ukazuje test správného spuštění aplikace, který spočívá v nastartování aplikace a ověření startu pomocí porovnání obrázků. Pokud test skončí korektně, funkce vrací True, v případě jakéhokoliv selhání False. Ostatní automatizované testovací případy mají obdobný formát, mají však vždy větší počet operací.

```
def 01_live_page(browser):
  Log.insert('{0} {1} 01 Live page'.format(browser.act_time(),
     browser.browser_type))
  # path to the expected result for image compare
  expected = '{0}_{1}_expected_result.png'.format('01_live',
     browser.browser_type)

  if browser.start_app():
    if browser.player_startup_screenshot(expected):
      browser.close_quit_button(feedback=True)
      browser.close_skip_feedback()
      return True
    else:
      # when images have different resolutions
      return False
  return False
```

Výpis 3: Nastartování aplikace a ověření startu

Funkce z kódu sady testů potom spouštím ve spouštěči sadu testů, kde definuji, jaké testy a s jakými parametry poběží, a jaká bude výchozí akce po dokončení každého testu. Zde také počítám statistiky, a v případě neúspěchu dojde po doběhnutí celé sady k odeslání emailu.

6.5 Lokalizace elementů

Lokalizace elementů je v Selenium možná více způsoby:

- atribut ID
- atribute class
- atribut name
- XPath selektor
- CSS selektor

- a jiné

Jak můžete vidět v předchozích ukázkách, rozhodl jsem se využívat CSS selektory. CSS selektory umožňují hledání podle libovolných atributů (čili v sobě zahrnují rovněž lokaci dle ID, class a name). Rovněž nejsem omezen hledáním dle jednoho atributu, můžu elementy a atributy za sebe řetězit tak, jak je povoleno v referencích CSS selekttorů [9]. Velice podobnou funkcionalitu poskytuje i XPath selektor [10]. Ten sice přidává možnost vyhledávání dle textového obsahu elementu, ale při jeho použití může docházet k různým výsledkům v odlišných prohlížečích a situacích, jelikož podpora XPath je v každém prohlížeči mírně odlišná [11] [12]. Lokalizace elementů dle CSS je ve všech prohlížečích stejná (za předpokladu, že je ve všech prohlížečích stejný HTML dokument).

6.6 Problémy

Při automatizaci se vyskytlo několik problémů, které jsem musel vyřešit.

Podpora Selenium Webdriver napříč prohlížeči se různí. Selenium Webdriver je oficiálně podporován pouze pro prohlížeč Mozilla Firefox, ostatní prohlížeče vyžadují doplňkový soubor, tzv. driver, který se stará o ovládání prohlížeče. Bohužel vývoj těchto driverů je na různých úrovních, tudíž má každý jiná omezení a jiné (zatím) nepodporované funkce.

Podpora funkce implicitly_wait je jednou z nich, není implementována pro prohlížeč mimo Firefox. Pokud je vyžadována interakce s elementem, který ještě na stránce není k dispozici, ať už tím, že ještě není stránka úplně načtena, nebo případně ještě nedoběhly asynchronní javascriptové funkce, které prvek zobrazí, je uživateli vrácena okamžitě výjimka. Tato funkce globálně říká prohlížeči: "Pokud při vykonávání nějaké události nenajdeš používaný element, je možné, že se do nastaveného timeoutu zobrazí."To je obzvláště výhodné v HTML5 aplikacích (jako je Frame), kde je JavaScriptu využíváno hojně.

Řešení této situace je ale i v ostatních prohlížečích možné, a to za použití funkce WebDriverWait, která zajistí čekání, dokud není splněna libovolná podmínka nebo nevypršel nastavený timeout:

```
WebDriverWait(browser, 10).until(
    browser.find_element_by_css_selector('div#dialog').is_displayed())
```

Výpis 4: Náhrada implicit_wait pro prohlížeče mimo Firefox

Tato funkce počká, dokud není zobrazen element div s atributem id=dialog. Počká však maximálně 10 vteřin, po uplynutí této doby je uživateli vrácena výjimka TimeoutException. Nevýhodou této funkce je jednak to, že je nutné funkci použít pro každý

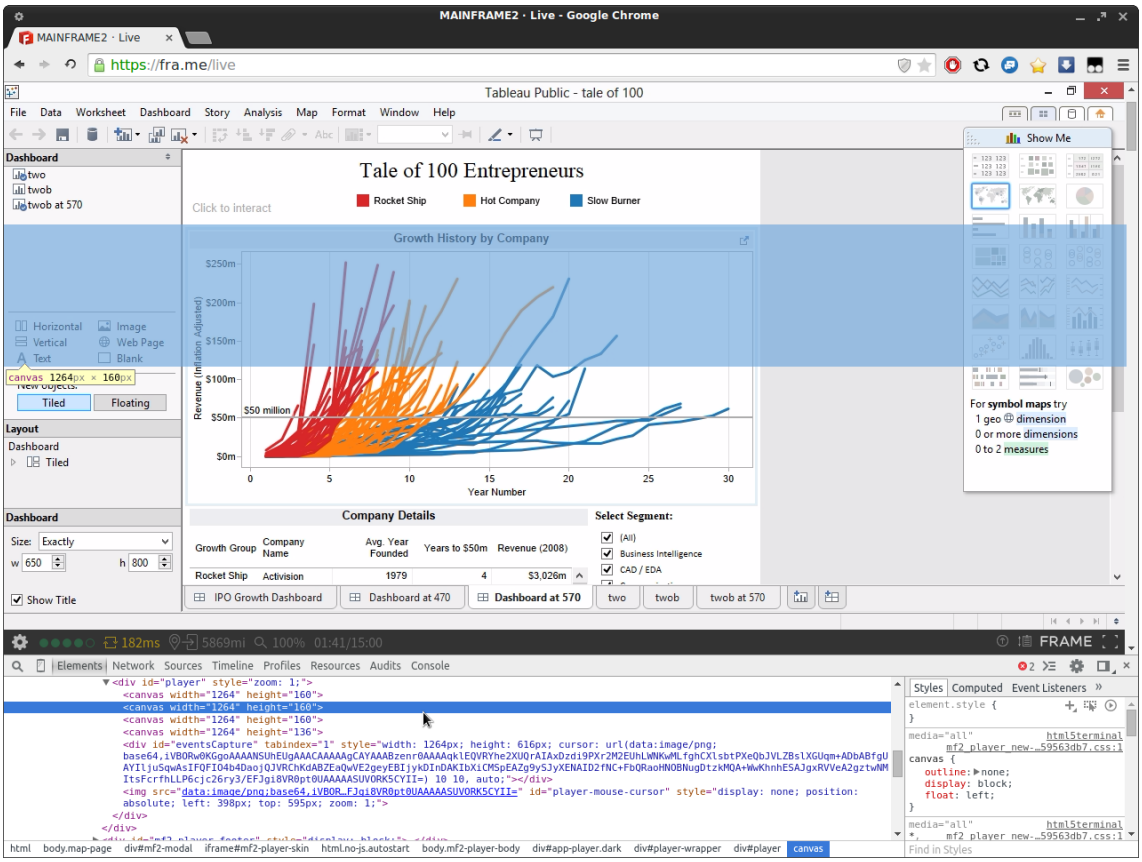
příkaz, u kterého by mohlo nutné čekání nastat, ale také nutnost odchytávání TimeoutException, aby byla v případě neznámého problému vrácena vhodná zpráva, standardně je totiž vrácena TimeoutException s prázdnou zprávou.

Podpora modálních dialogů je implementována jak ve Firefoxu, tak i Google Chrome. Driver pro prohlížeč Safari je však implementován v JavaScriptu, který běží na stejné úrovni jako modální dialogy. Tím vzniká problém, kdy nad nimi driver není schopen převzít kontrolu. Všechny dialogy mají standardně nastavenou akci na zamítnutí těchto dialogů, což nebylo v mém případě žádoucí. Z tohoto důvodu byla dočasně vyloučena podpora pro Safari. Dočasně proto, že jeden z cílů Frame je také nahrazení modálních dialogů svými vlastními dialogy, napsanými v HTML5. Pak budou testy fungovat bez problémů i pro Safari.

Podpora Internet Exploreru při automatických testech v době psaní této práce není funkční, ani není prioritou. Při používání Selenium Webdriver s Internet Explorer dochází při manipulaci se stránkou k náhlému ukončení spojení mezi Selenium Webdriver a driverem pro Internet Explorer. Hledání příčiny ukončení spojení bylo Framem odloženo do budoucna s nízkou prioritou.

7 Ověření WebGL obsahu

Ověření startu aplikace bylo jedním z prvních a zároveň jedním z nejdůležitějších úkolů, které jsem musel při automatizaci vyřešit. Je totiž nutné zjistit, zda byla aplikace korektně nastartovaná a uživatel ji může začít používat. Aplikace je zobrazována uživateli za použití HTML5 technologií, konkrétně WebGL a canvas, jak je patrné z obrázku 4 na straně 24. Zde však nastává problém, jak ověřit správný obsah canvas elementu, který je vykreslován jako jedna bitmapa. Podle materiálů, které jsem našel a měl k dispozici není jiná možnost, která by byla dostačující, než porovnávání obrázků.



Obrázek 4: Použití canvas technologie

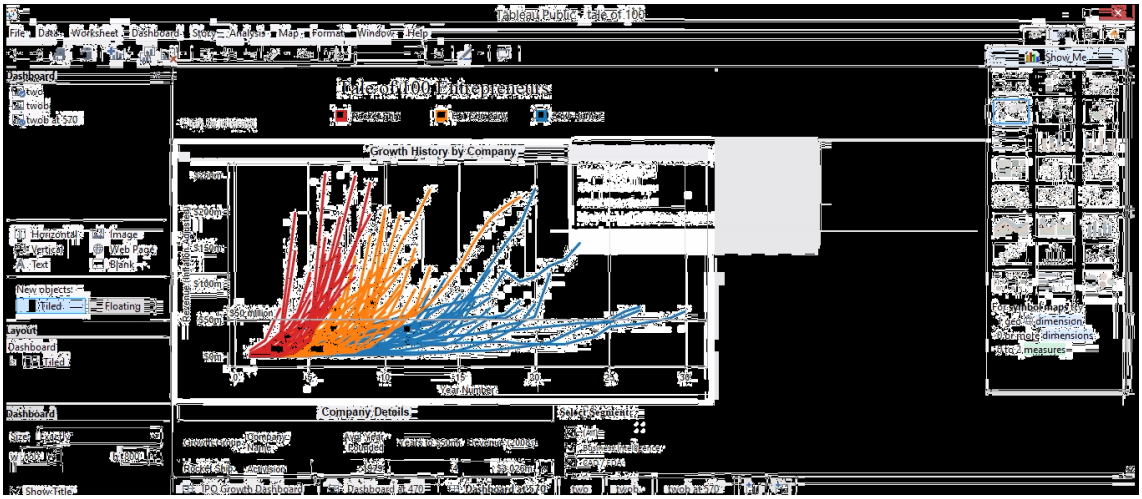
Obrázky načítám za pomocí knihovny PIL [13] a samotné porovnávání probíhá tak, že každý bod jednoho obrázku porovnám s bodem na stejných souřadnicích druhého obrázku. Pokud se jedná o barevný obrázek, je nutné počítat s tím, že každý bod je složen ze 3 barevných složek (RGB). Jednotlivé barevné složky nabývají hodnot 0 až 255. Rozdíl mezi hodnotami stejné barevné složky postupně přičítám do připravené proměnné zpočátku definované na 0, kterou nakonec vydělím maximální hodnotou možného rozdílu,

což je 255. Výsledek je nutné vydělit počtem barevných složek v obrázku, a tím získám procentuální rozdíl mezi obrázky ve tvaru desetinného čísla.

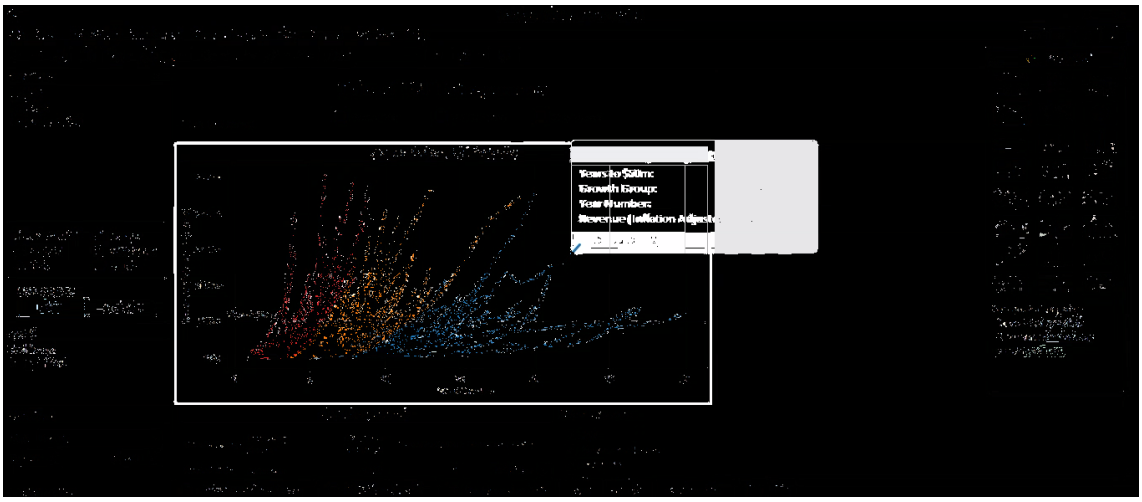
Nevýhodou tohoto způsobu porovnávání je fakt, že je nutné mít jak stejný typ obrázku, tak i naprosto stejné rozměry. S tím by však v rámci mého testování problém být neměl, zvolil jsem proto tuto jednoduchou metodu.

Pro automatické testy jsem stanovil jako akceptovatelný rozdíl do 5%. Pokud je větší, je vyhodnocen start aplikace jako neúspěšný. Tuto hranici jsem nastavil dle výsledků získaných otestováním rozdílů několika různých aplikací za různých podmínek, zejména rozlišení. Při úspěšném nastartování aplikace rozdíl mezi očekávaným a aktuálním výsledkem nepřesahoval 3%. Hranice se dá samozřejmě kdykoli změnit v konfiguračním souboru sady testů.

Procentuální podobnost je důležitý údaj, může však být složité na první pohled odhadnout, co přesně se na obrázcích změnilo. Proto při výpočtu podobnosti rovnou připravuji ještě rozdílový obrázek, na kterém půjdou rozdíly rozeznat okamžitě. Rozdílový obrázek je vytvořen tak, že stejné pixely se změní na černou barvu a tam, kde je identifikován rozdíl, se promítne pixel z porovnávaného obrázku. Nutno však mít na paměti, že kvalita streamu aplikace může vzhledem k internetovému připojení kolísat a Frame automaticky sníží kvalitu streamu. Tento fakt se rovněž projeví při porovnávání. Citlivost tvorby rozdílového obrázku je standardně nastavena tak, aby jakákoli změna byla vykreslena. Příklady takového výstupu jsou viditelné na straně 26 na obrázku č. 5, sníženou citlivost zase ukazuje obrázek č. 6. Mírně snížená citlivost vyfiltruje téměř nepozorovatelné rozdíly a umožní testerovi či podpůrnému týmu vidět opravdový problém, v testovaném případě se jednalo o zobrazený tooltip v aplikaci.



Obrázek 5: Rozdílový obrázek s maximální citlivostí



Obrázek 6: Rozdílový obrázek se sníženou citlivostí

8 Plány do budoucna

Automatizace testů bude dále vyvíjena, sám již nacházím oblasti, ve kterých bych viděl možnosti zlepšení. Jedno takové vylepšení bude přesunutí CSS selektorů ze stávajících testů do databáze. V případě změn v HTML struktuře je nutné projít kód a provést změny v každém selektoru, kterého se změna týká. Pokud budou tyto selektory uloženy v centrální databázi, bude tato editace značně jednodušší a rychlejší.

9 Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné v průběhu odborné praxe

Produkt je postaven na technologiích HTML5, o kterých jsem dostal dobré povědomí v předmětu Vývoj internetových aplikací. Díky tomu jsem hned věděl, že při automatizaci nebude možné lokalizovat nějakou část uvnitř canvas elementu běžící aplikace, protože že se neskládá z vrstev, ale je vykreslen pouze jako jedna bitmapa. Tato znalost mi umožnila lépe vybrat metodu automatizace.

Celá automatizace byla napsána v jazyce Python, se kterým jsem se setkal v předmětu Skriptovací programovací jazyky a jejich aplikace, jehož obsah mi plně dostačoval pro všechny základní operace. V tomto předmětu jsme také využívali externích knihoven, instalace a přilinkování Selenium Webdriver tedy byla otázka okamžiku. Také jsme dodržovali Python konvence, které dodržují Python vývojáři po celém světě, v syntaxi Selenia jsem se díky tomu orientoval velice rychle. Vývoj probíhal na operačním systému GNU/Linux, kde jsem uplatnil vědomosti získané z předmětu Správa operačních systémů. Především se jedná o správu samotného systému, kdy jsem byl schopen rychle nastavit systém tak, aby mi ulehčil práci.

Komunikace se zákazníkem probíhala pouze v angličtině, ve které jsem se značně zlepšil v průběhu čtyř semestrů (Ab/I-FEI - Ab/IV-FEI).

10 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Mezi nejvíce scházející znalosti či dovednosti bych zařadil nutnost práce se softwarem určeným k verzování, zákazník zvolil technologii git. Značnou chvíli zabralo, než jsem pochopil manipulaci s různými větvemi stejného kódu, spojování těchto změn, a než jsem byl schopen nastavit git na lokálním počítači. Myslím, že verzovací systémy se používají v drtivé většině firem, a tudíž by pode mého názoru bylo její zařazení do studijního plánu pro studenty velice přínosné.

11 Závěr

Praxe mi umožnila seznámit se s procesem testování softwarových produktů, zejména s různými technikami a přístupy a jejich aplikací v procesu vývoje softwaru. Osvojil jsem si proces testování od návrhu testovacích případů přes průběžné manuální testování až po automatizaci testů. Zároveň se značně změnil můj pohled na testování softwarových produktů. Doposud jsem si myslel, že testovací týmy průběžně stereotypně manuálně testují produkt. Praxe mi však ukázala, že manuální testování je pouze část celku.

Dále mi praxe umožnila seznámení se s technologiemi pro automatizaci testů, zejména však s technologií Selenium Webdriver a možnostmi jejího použití. Získané znalosti využila ve firmě řada spolupracovníků, kterým jsem ušetřil značné množství času získanými postřehy, jejichž absence mě v začátcích také zpomalovala. Zejména se jedná o informace o různé podpoře mezi prohlížeči a o funkcích automatického čekání, pokud element ještě není k dispozici.

Mnou vyvinutá automatizace se osvědčila a je spouštěna cronem několikrát denně na odděleném serveru. Tím, že jsem vytvořil oddělené sady testů, jsou kritické testy spouštěny výrazně častěji, například každých 10 minut, než jiné, u kterých není taková četnost nutná, a kde stačí dvouhodinový interval. Odeslání emailového upozornění v případě neúspěchu může být ve Framu napojeno kupříkladu na upozornění textovou zprávou na mobilní telefon. Podpůrný tým ve Framu je okamžitě informován a může problém řešit dříve, než se projeví u jejich zákazníků.

12 Reference

[1] profiq s.r.o. *profiq* [online]. © 2012 [cit. 2015-03-20]. Dostupné z: http://www.profiq.com

[2] Frame: Run any software in a browser *Frame* [online]. © 2015 [cit. 2015-05-01]. Dostupné z: http://fra.me

[3] Certifikovaný tester: Učební osnovy pro základní stupeň *International Software Testing Qualifications Board* [online]. 15.6.2013 [cit. 2015-03-20]. Dostupné z: http://castb.org/wp-content/uploads/2013/11/ISTQB-CTFL-Syllabus-v2011-CZ-Beta1.pdf

[4] Selenium Documentation *SeleniumHQ* [online]. 27.4.2015 [cit. 2015-03-20]. Dostupné z: http://www.seleniumhq.org/docs/index.jsp

[5] HtmlUnit *Gargoyle Software Inc.* [online]. 20.4.2015 [cit. 2015-04-18]. Dostupné z: http://htmlunit.sourceforge.net/

[6] Python 2.7 documentation *Python Software Foundation* [online]. 20.4.2015 [cit. 2015-04-18]. Dostupné z: https://docs.python.org/2/

[7] Watir WebDriver *Alister Scott* [online]. [2013] [cit. 2015-04-18]. Dostupné z: http://watirwebdriver.com/

[8] Squish GUI Tester *froglogic* [online]. © 2015 [cit. 2015-04-18]. Dostupné z: http://www.froglogic.com/squish/gui-testing/

[9] Selectors Level 3 *World Wide Web Consortium (W3C)* [online]. 29.9.2011 [cit. 2015-04-18]. Dostupné z: http://www.w3.org/TR/css3-selectors/

[10] XPath Tutorial *w3schools.com* [online]. © 1999-2015 [cit. 2015-04-18]. Dostupné z: http://www.w3schools.com/xpath/default.asp

[11] Automated Web Testing: Traps for the Unwary with Simon Stewart *EuroSTAR Conferences* [online]. 28.1.2013 [cit. 2015-03-20]. Dostupné z: http://old.eurostarconferences.com/blog/2013/1/28/liveblogged!-automated-web-testing-traps-for-the-unwary-with-simon-stewart

[12] XpathInWebDriver *selenium* [online]. 14.1.2013 [cit. 2015-03-20]. Dostupné z: https://code.google.com/p/selenium/wiki/XpathInWebDriver

[13] Python Imaging Library Handbook *Fredrik Lundh* [online]. © 1995-2014 [cit. 2015-04-18]. Dostupné z: http://effbot.org/imagingbook/pil-index.htm